05

Operación de listado en el ProductoDAO

Transcripción

[00:00] Ahora que conocimos el patrón del diseño DAO y entendimos su responsabilidad, vamos a seguir refactorizando la clase productoController para mover toda la lógica de acceso a la base de datos para la clase productoDAO. La primera operación que vamos a mejorar va a ser la del método listar.

[00:16] La idea es mejorar este método para que, más allá de utilizar el producto DAO para acceder a la base de datos, nosotros también vamos a estar utilizando la clase de modelo producto para que tengamos una mejor representatividad del modelo en nuestro código.

[00:31] Lo primero que vamos a hacer aquí en el método listar va a ser instanciar un productoDAO. Vamos a hacer un productoDAO = new ProductoDAO y recibe en new ConnectionFactory().recuperaConexion(). Entonces aquí estamos instanciando el productoDAO que recibe una conexión de nuestro pool de conexiones. Pero si lo estamos instanciando aquí, vamos a estar repitiendo la operación que ya hicimos en el método guardar.

[01:08] Estamos instanciando el mismo objeto en varias partes de la clase de productoController. Entonces en lugar de instanciar productoDAO adentro de cada método del productoController, para estar repitiendo el código así, lo que vamos a hacer es convertir esta variable en un campo de la clase productoController.

[01:28] Y acá este productoDAO lo que vamos a hacer es moverlo de acá y lo vamos a llevar aquí hacia arriba. ¿Para hacer qué? Lo vamos a inicializar cuando nosotros inicialicemos el productoController. ¿Cómo se hace eso? En un constructor. Entonces aquí voy a crear un constructor de productoController, en donde cuando inicialicemos este objeto nosotros vamos a hacer un this.productoDAO = ProductoDAO, acá lo había copiado y ya lo tenía.

[01:57] new ProductoDAO con el connectionFactory().recuperaConexion(). Pero aquí una vez más, Eclipse se queja de SQLException. Llegamos a un punto en que tenemos que tratar esta situación de otra forma. No podemos más estar declarando que todos lo métodos lancen la SQLException.

[02:17] Aquí lo que está pasando es que, siempre que declaramos y nuestros métodos lanzan una excepción, nosotros delegamos la responsabilidad para que sea tratada en otro lugar. En nuestro caso la estamos tratando en ControlDeStockFrame. Acá, si miramos acá, en esta clase tenemos varios try catch, en donde estamos haciendo un catch de SQLException.

[02:40] Entonces eso está mal. No es una buena práctica el hacer eso, delegar las excepciones de una capa para otras. Lo ideal es que las tratemos en el mismo lugar en donde el error ocurre. Entonces, en este caso acá de connectionFactory ¿en donde tenemos que tratarla? Adentro de connectionFactory.

[03:00] Entonces vamos a entrar en el método recuperaConexion y en este comando acá de dataSource.getConnection(); que es donde ocurre la posible excepción, nosotros la vamos a agregar dentro de un bloque de try catch, igual como estamos haciendo en el ControlDeStockFrame, nosotros vamos a hacer un throw new RuntimeException.

[03:26] Ahí está. Y vamos a encapsular la excepción original y así tenemos una excepción no chequeada que podemos lanzar sin ningún problema. Va a ocurrir un error, pero por lo menos nosotros no tenemos que estar propagando

esa declaración de que este método lanza una excepción y eso va subiendo hasta el primer lugar que lo llama.

[03:49] La idea es que con eso, de a poco vayamos sacando el throws SQLException de los otros métodos también. Acá podemos hacer lo mismo en productoDAO por ejemplo. Aquí, este throws SQLException de guardar nosotros podemos sacarlo y en ese try de acá, el primero, este exception, nosotros lo vamos a cambiar para SQLException, ahí está mejor.

[04:22] Y nosotros vamos a sacar este control también, de la transacción que ya vimos que funciona. La vamos a sacar, y aquí en lugar de estar haciendo un print stack trace o row back, vamos a estar haciendo otra vez el throw new RuntimeException(e); ahora sí, con la excepción encapsulada también. Listo.

[04:49] Aquí en productoController, en el método guardar, nosotros podemos sacar este throws SQLException y por consecuencia, en el ControlDeStockFrame, tenemos que venir al método de guardar también y sacar este bloque de try catch porque no es más necesario. Ahora sí sacamos la responsabilidad del ControlDeStockFrame para el método de guardado. Listo.

[05:15] Ahora vamos a volver aquí en el método de listar del controller. Y la idea es que nosotros tomemos el atributo de productoDAO que acá ya lo podemos sacar, de guardar, ya no necesita más y nosotros vamos a tomar el product DAO aquí, vamos a hacer un productoDAO.listar(); entonces este productoDAO tiene que devolvernos este listado de productos que el controller estará haciendo ahí.

[05:49] Entonces esta lógica de acá la vamos a mover para adentro del método listar de productoDAO. Ahora entonces vamos a, primero sacar este SQLException y todo este pedazo de código que tenemos aquí en el controller, nosotros lo vamos a cortar y mover para el nuevo método de productoDAO que vamos a crear ahora, haciendo command o "Ctrl + 1" y creamos el método en la clase productoDAO.

[06:15] Aquí voy a borrar esta parte y vamos a pegar el método aquí, está bien ahí. Ahora lo que vamos a hacer, ya tenemos aquí esta parte, esto acá lanza un SQLException. ¿Qué vamos a hacer? catch (SQLException e) y hacemos un throw new RuntimeException(e); con la excepción original encapsulada. Ahora, lo que tenemos que hacer es cambiar este map para el producto, para que tengamos la representatividad que habíamos dicho.

[06:56] Ya realizamos la primera parte que era mover la lógica de acceso a los datos para el productoDAO y ahora vamos a la segunda parte, en donde vamos a convertir el map para el producto. Empezando aquí, ya vamos a, en la declaración, a hacer un producto acá. Ya hacemos este cambio, en este list de resultado hacemos lo mismo también y acá en el map, en donde estamos mapeando el resultado para agregar al listado, nosotros vamos a hacer un producto.

[07:30] Podemos mantener el nombre fila, no hay problema, pero aquí, en lugar de estar haciendo un fila.set cada campo, vamos a crear el constructor de producto y vamos a estar recibiendo todos los valores que ya tenemos. Entonces, este ID es un entero, ya no necesitamos más enviar como un string, el próximo campo es el nombre ya lo podemos sacar también y agregarle el constructor.

[07:56] Acá estoy moviendo los valores. Pero si quieren pueden escribir también, pero aquí estamos haciendo ahora la descripción y por último la cantidad. Vean que estoy agregando los valores así como son: Int, string, string, Int otra vez. No estoy más convirtiendo todo para un string porque el producto tiene sus atributos con sus tipos ya bien declarados.

[08:25] Entonces esta parte de fila.put podemos sacar todo y por último, vamos a crear este constructor acá en producto. Ahí está. Vamos a cambiar los nombres. Acá los nombres son id, nombre, descripción y por último cantidad. Ahora vamos a asignar todos los valores, this.id = id; this.nombre = nombre; this.descripcion = descripcion; this.cantidad = cantidad; ahí está.

[09:09] ¿Ahora qué falta? Falta aquí en productoController cambiar este retorno para que sea un listado de producto. Ahora sí. Ya lo tenemos pero aún hay un errorcito aquí, vamos a ver qué pasa. No pasa nada más. Faltó solamente guardar el productoDAO, ahí está. Ahora sí, y por último en el ControlDeStockFrame, vamos al método de listado para cambiar el resultado de productos para.

[09:40] Acá podemos sacar también ya este try catch. Miren qué bueno eso. Este productos podemos hacer un elegí todo y puse, listo, var productos = this.productoController.listar(); y ete var en el Java 11 ya dinámicamente en scopes locales, ya en la asignación, nosotros cuando declaramos una variable Java tenemos que inicializar con su valor.

[10:10] En la inicialización, la JVM ya sabe cual es su tipo. Entonces, en este caso el var de productos va a ser del tipo list de productos. Acá por último, tenemos los gets del Map que tenemos que convertir para los getters que tenemos en la propia clase de producto. Entonces getId, getNombre, getDescripcion y el último acá getCantidad.

[10:41] ¿Qué falta? Falta crear el getId, entonces aquí venimos, hacemos un getId de Integer y hacemos un return this.id, pero para dejar mas organizado acá el código, voy a dejarlo junto de su set, así tenemos todo bien ordenado. Guardé acá los cambios, hice un organize imports y ahora vamos a probar.

[11:14] Listo. Todo sigue funcionando igual y nuestra aplicación va quedando cada vez más linda y completa, siguiendo buenas prácticas de programación y estándares de desarrollo del mercado.

[11:25] Vamos incrementando cada vez más la capa de persistencia y acceso a los datos acá en el productoDAO. Y las próximas operaciones que vamos a hacer son las de modificación y exclusión de un registro, que están en productoController todavía. Así nosotros vamos a completar todas las

operaciones de ABM, que son alta, baja y modificación, y también la del listado.

[11:48] Ahora cada capa de aplicación tiene su responsabilidad bien definida. La capa DAO acá se encarga de mantener toda la lógica de acceso a la base de datos, las queries y traducción de result set para la clase de producto que es nuestro model, y la capa de controller se encarga de llamar las demás clases para completar la operación solicitada por la view.

[12:11] Y por último, la view, que es la que hace las requisiciones para el controller, para hacer todo esto que acá es nuestra pantallita del listado de productos y con el formulario que es la representación de nuestra clase ControlDeStockFrame. Todo esto que acabo de comentar es un patrón de diseño conocido como MVC, Model View Controller.

[12:35] Pero este tema voy a contar un poco más en la próxima clase. Por ahora quedamos por aquí y les dejo como desafío refactorizar las demás operaciones de productoController, que son las de eliminar un registro y de modificar un registro de producto siguiendo todo lo que aprendimos hasta aquí. Nos vemos en la próxima clase.